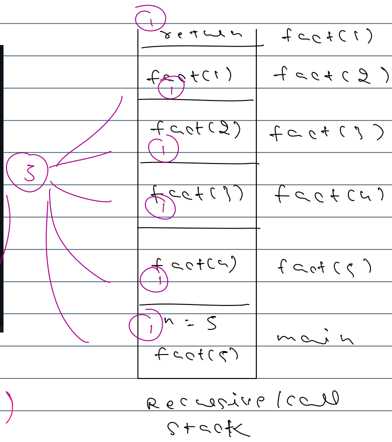
Recursive Time complexity

→ one fn call ⇒ factorial

→ multiple fn call ⇒ fibonacci

one fn call

```java
public class Factorial {
    public static void main(String[] args) {
        int n = 5;
        int factorial = fact(n);
        System.out.println(factorial);
    }

    public static int fact(int n) {
        if (n == 1)
            return 1;
        return n * fact(n - 1);
    }
}
```

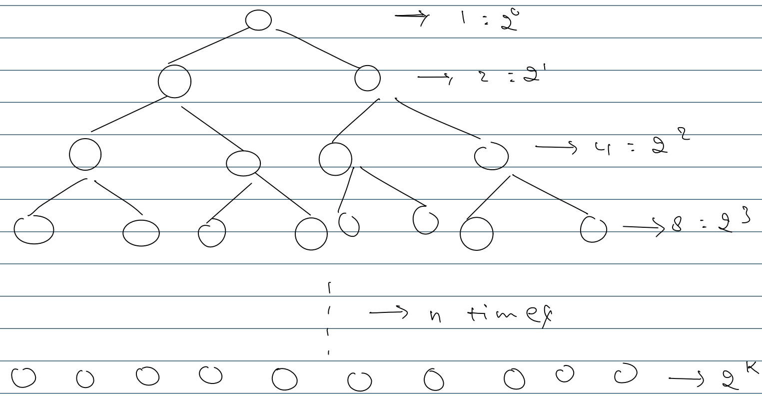


n times (recursive frames)

$\boxed{O(n)}$

multiple fn calls

```java
public class Fibonacci {
    public static void main(String[] args) {
        int n = 5;
        System.out.println(fib(n));
    }

    public static int fib(int n) {
        if (n == 0 || n == 1) {
            return n;
        }
        return fib(n - 1) + fib(n - 2);
    }
}
```



$(2^0 + 2^1 + 2^2 - - - - 2^n) \times 1$ → Geometric progression ⇒ $\frac{a.(r^n - 1)}{(r-1)}$

$a = 2^0$

$r = 2$

a = first term

r = difference

$\frac{2^0 \times (2^{n-1} - 1)}{2-1} = 1.(2^{n-1} - 1)$

$= 2^{n-1} - 1 = 2^{n-1} - 1$

→ Time complexity ⇒ $O(2^n)$

Recurrence Relation

fn of times which helps in calculating time complexity.

```java
public class Factorial {
    public static void main(String[] args) {
        int n = 5;
        int factorial = fact(n);
        System.out.println(factorial);
    }

    public static int fact(int n) {
        if (n == 1)
            return 1;
```

$f(n) = f(n-1) + 1$

$\boxed{T(n) = T(n-1) + 1}$

Base case

↓

$T(0) / T(1) = 1$

```java
public class Factorial {
    public static void main(String[] args) {
        int n = 5;
        int factorial = fact(n);
        System.out.println(factorial);
    }

    public static int fact(int n) {
        if (n == 1)
            return 1;
        return n * fact(n - 1);
    }
}
```

$f(n) = f(n-1) + 1$

$\boxed{T(n) = T(n-1) + 1}$

Base case

$T(0) \mid T(1) = 1$

Elimination method

$T(n-1) = T(n-2) + 1$

$T(n) = T(n-1) + 1 \rightarrow$ 1st

$= [T(n-2) + 1] + 1$

$T(n-2) = T(n-3) + 1$

$= T(n-2) + 2 \rightarrow$ 2nd

$= [T(n-3) + 1] + 2$

$= T(n-3) + 3 \rightarrow$ 3rd

$\rightarrow$ after k expansions

$T(n) = T(n-k) + k \rightarrow$ kth

Base case

input size is 1

$n - k = 1$

$\boxed{k = n-1}$

$T(n) = T(1) + (n-1)$

$T(n) = c + (n-1)$

$T(n) = (c-1) + n$

$T(n) = c + n$

$\boxed{T(n) = O(n)}$

Binary Search

$\boxed{T(n) = T\left(\frac{n}{2}\right) + 1}$

$T(n/2) = T(n/4) + 1$

$T(n) = T(n/2) + 1 \rightarrow$ 1st

$T(n) = [T(n/4) + 1] + 1$

$T(n) = T(n/4) + 2$

$T(n/4) = T(n/8) + 1$

2nd

$T(n) = [T(n/8) + 1] + 2$

$T(n) = T(n/8) + 3 \rightsquigarrow T(n) = T(n/2^3) + 3 \rightarrow$ 3rd

$\rightarrow$ after k expansions

$T(n) = T(n/2^k) + k \rightarrow$ kth

Base case:-

input size is 1

$\frac{n}{2^k} = 1$

$n = 2^k$

$\log_2 n = \log_2 2^k$

$\log_2 2^k = \log_2 n$

$k \cdot \cancel{\log_2 2} = \log_2 n$

$\boxed{k = \log_2 n}$

$T(n) = T(1) + \log_2 n$

$T(n) = c + \log_2 n$

$\boxed{T(n) = \log_2 n}$

merge Sort

$$\boxed{T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + n}$$

$$\boxed{T(n) = 2 \cdot T(n/2) + n}$$

$T(n/2) = 2 \cdot T(n/4) + n/2$

$T(n) = 2 \cdot T(n/2) + n$ → 1st

$= 2[2 \cdot T(n/4) + n/2] + n$

$= 4 \cdot T(n/4) + \cancel{2} \cdot \frac{n}{\cancel{2}} + n$ → 2nd

$= 4 \cdot T(n/4) + 2n$ ⟶ $2^2 \cdot T(n/2^2) + 2n$

$T(n/4) = 2 \cdot T(n/8) + n/4$

$= 4[2 \cdot T(n/8) + n/4] + 2n$

$= 8 \cdot T(n/8) + \cancel{4} \cdot \frac{n}{\cancel{4}} + 2n$ → 3rd

$= 8 \cdot T(n/8) + 3n$ ⟶ $2^3 \cdot T(n/2^3) + 3n$

→ after k expansions

$$\boxed{T(n) = 2^k \cdot T\left(\frac{n}{2^k}\right) + k \cdot n}$$ → kth

Base condition

input size is 1

$\frac{n}{2^k} = 1$

$\boxed{n = 2^k}$

$\boxed{k = \log_2 n}$

$T(n) = 2^{\log_2 n} \cdot T(1) +$

$\log_2 n \times n$

$T(n) = n \cdot C + \log_2 n \cdot n$

$$\boxed{T(n) = \log_2 n \cdot n}$$

Fibonacci

$$\boxed{T(n) = T(n-1) + T(n-2) + 1}$$

$T(n) = [T(n-2) + T(n-3) + 1] + [T(n-3) + T(n-4) + 1] + 1$

$= T(n-2) + 2 \cdot T(n-3) + T(n-4) + 3$

relation keeps growing

$T(n-1) + T(n-2) + 1 \leq T(n-1) + T(n-1) + 1$

$T(n-1) = 2 \cdot T(n-2) + 1$

$T(n) = 2 \cdot T(n-1) + 1$ → 1st

$= 2[2 \cdot T(n-2) + 1] + 1$

$T(n-2) = 2 \cdot T(n-3) + 1$

$= 4 \cdot T(n-2) + 3$ ⟶ $2^2 \cdot T(n-2) + 3$

$= 2[2 \cdot T(n-2) + 1] + 1$

$= 4 \cdot T(n-2) + 3 \rightsquigarrow 2^2 \cdot T(n-2) + 3$ — 2nd

$T(n-2) = 2 \cdot T(n-3) + 1$

$= 4[2 \cdot T(n-3) + 1] + 3$

$= 8\,T(n-3) + 7 \rightsquigarrow 2^3 \cdot T(n-3) + 7$ — 3rd

$\rightarrow$ after k expansions

$$\boxed{T(n) = 2^k \cdot T(n-k) + k}$$

k → multiple of k → $2^k - 1$ — kth

Base condition

input size is 1

$n - k = 1$

$\boxed{k = n-1}$

$T(n) = 2^{n-1} \cdot T(1) + 2^{n-1} - 1$

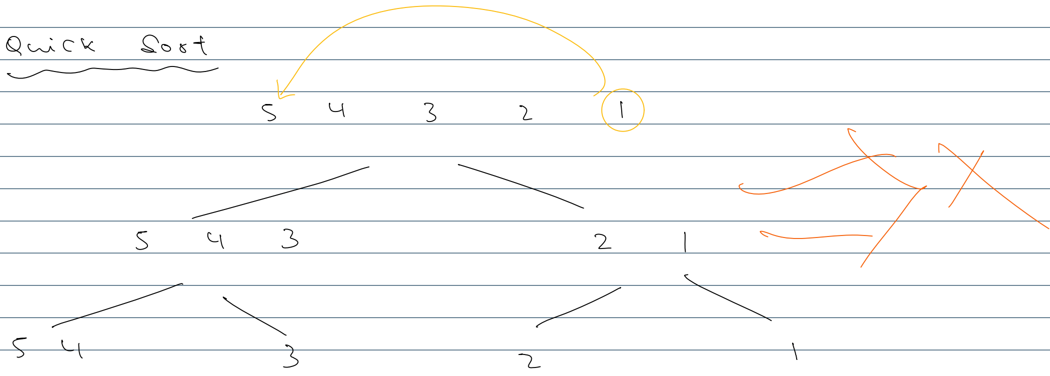
$T(n) = 2^{n-1} \cdot c + 2^{n-1} - 1$

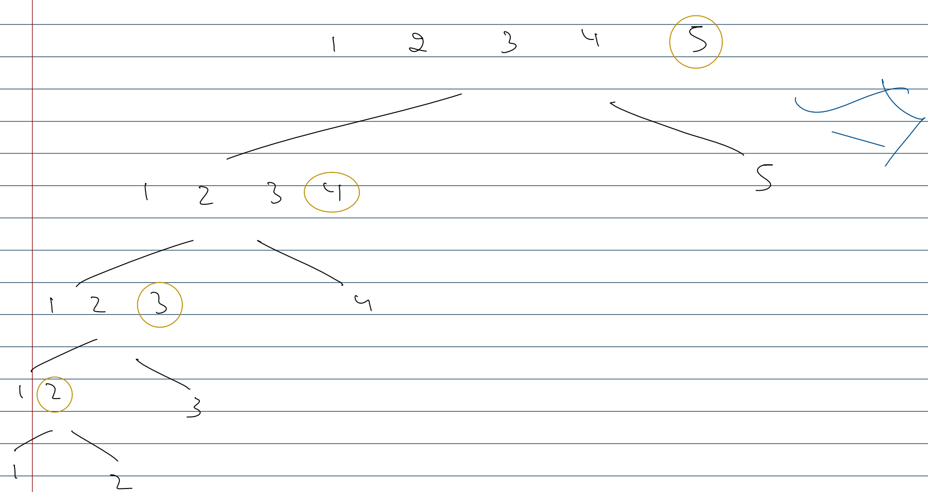
$T(n) = \frac{2^n}{2} \cdot c + 2^{n-1} - 1$

$T(n) = 2^n + 2^{n-1} - 1$

$\boxed{T(n) = O(2^n)}$

Quick Sort





$T(n) = T(n-1) + n$

$\Downarrow$

$\boxed{T(n) = O(n^2)}$

↓

worst

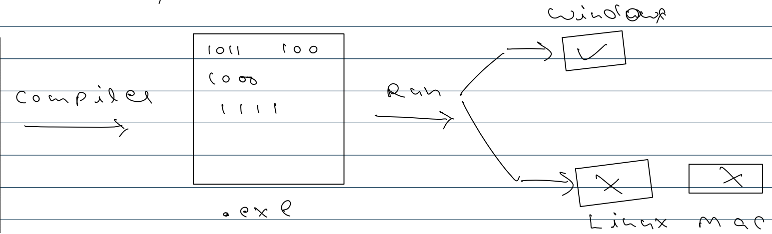
```
public class Main {
  public static void main(String[] args) {
    int[] arr = { 5, 7, 2, 1, 8, 3, 4 };
```

```
        sort(arr, 0, arr.length - 1);
            for (int i = 0; i < arr.length; i++) {
                System.out.print(arr[i] + " ");
            }
    }

    public static void sort(int[] arr, int si, int ei) {
        if (si >= ei) {
            return;
        }
        int idx = partition(arr, si, ei);
        sort(arr, si, idx - 1);
        sort(arr, idx + 1, ei);

    }

    public static int partition(int[] arr, int si, int ei) {
        Random rn = new Random();
        int ri = rn.nextInt(ei - si) + si;

        int tt = arr[ei];
        arr[ei] = arr[ri];
        arr[ri] = tt;

        int item = arr[ei];
        int idx = si;
        for (int i = si; i < ei; i++) {
            if (arr[i] <= item) {
                int temp = arr[i];
                arr[i] = arr[idx];
                arr[idx] = temp;
                idx++;
            }

        }
        int temp = arr[ei];
        arr[ei] = arr[idx];
        arr[idx] = temp;
        return idx;
    }

}
```

OOPS

windows

```
#include <iostream>
using namespace std;
int main() {
    cout<<"Hello World!";
}
```

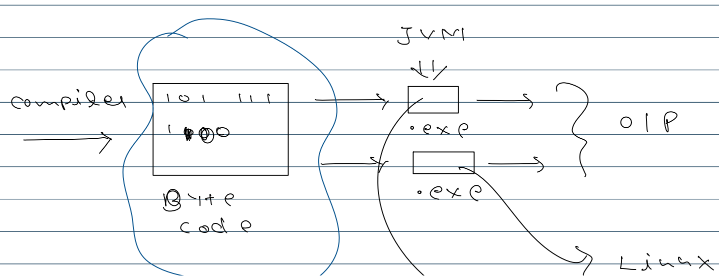


source code

C / C++ not platform independent

windows

```
public class Main {
    public static void main(String args[]) {
        int a = 5;
        int b = 7;
        int c = a + b;
        System.out.println("Hello World!");
    }
}
```

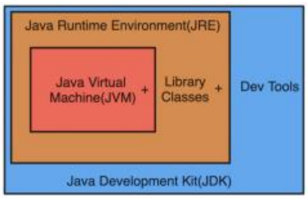


```
    }
}
```

Byte code

.exe → Linux → mac

→ source code



Why OOPS ?
→ maintenable
→ scalable
→ reusable
→ clean
→ modular

Programming paradigm
↓
how to divide a s/w into smaller parts

→ Procedural (Sequential) ⇒ 1 2 3

→ Object -Oriented (Data and code)

OOPS → Real world entities
→ Attributes (Properties)
→ Methods (Actions)

Akash
→ Properties ⇒ name, age, height
→ Actions ⇒ walk( ), talk( ), teach( )

Object → when we combine data and functions, that operate on the data, they form object.

class → It is the blueprint of object.
Object → An object is an instance of a class.

class → Human
↓
Object
→ Akash
→ Ayush
→ Abhay

```java
public class Student {
    2 usages
    String name;
    2 usages
    int age;

    public void introduceYourSelf() {
        System.out.println("My name is " + name + " age is " + age);
    }
}
public class StudentClient {
    public static void main(String args[]) {

        System.out.println("Hello Akarsh");
        Student s = new Student();

        System.out.println(s.age);
        System.out.println(s.name);
    }
}
```

JVM